Exercise 2 — Solution

Tools

Online notebook: `https://colab.research.google.com/`

Coding in **Python** (NumPy + Matplotlib).

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
```

Board / Paper Tasks

Example 1: Manual 8-point FFT (DIT Algorithm)

Compute the 8-point **Decimation-in-Time FFT** of the sequence

$$x[n] = [\,1,\ 0,\ 1,\ 0,\ -1,\ 0,\ -1,\ 0\,].$$

Tasks:

1. Write down the *bit-reversal order* for $N = 8$ and reorder $x[n]$ accordingly.
2. Perform three FFT stages (butterflies) using the radix-2 DIT algorithm.
3. Show the twiddle factors used in each stage:
$$W_N^k = e^{-j2\pi k/N}.$$
4. Compute the final spectrum $X[k]$ (use approximate numerical values for the complex numbers).

```
# ---------------------------------------------------------------
# Example: 8-point FFT of x = [1,0,1,0,-1,0,-1,0]
# ---------------------------------------------------------------
x = np.array([1,0,1,0,-1,0,-1,0], dtype=complex)
N = len(x)
W8 = np.exp(-1j * 2 * np.pi / N)

print("Input x[n] =", x)

# ---------------------------------------------------------------
# Reorder:
# ---------------------------------------------------------------
def bit_reversed_indices(N: int):
    m = int(np.log2(N))
    assert 2**m == N, "N must be a power of two"
```

```python
    idx = np.arange(N)
    # Reverse m bits of each index
    br = np.zeros(N, dtype=int)
    for i in range(N):
        b = format(i, f'0{m}b')[::-1]    # binary, reversed
        br[i] = int(b, 2)
    return br

perm = bit_reversed_indices(N)
x0 = x[perm]    # bit-reversed input

print("\nReorder:", x0)

# -------------------------------------------------------------
# Stage 1: 2-point butterflies (no twiddles)
# -------------------------------------------------------------
x1 = np.zeros(N, dtype=complex)
for k in range(0, N, 2):
    a, b = x0[k], x0[k+1]
    x1[k]   = a + b
    x1[k+1] = a - b
print("\nStage 1:", x1)

# -------------------------------------------------------------
# Stage 2: 4-point butterflies, twiddles W8^0 and W8^2 = -j
# -------------------------------------------------------------
x2 = np.zeros(N, dtype=complex)
for k in range(0, N, 4):
    a0, a1, a2, a3 = x1[k:k+4]
    x2[k]   = a0 + a2
    x2[k+1] = a1 + (W8**2) * a3    # multiply by -j
    x2[k+2] = a0 - a2
    x2[k+3] = a1 - (W8**2) * a3
print("\nStage 2:", np.round(x2, 4))

# -------------------------------------------------------------
# Stage 3: 8-point butterflies, twiddles W8^0..W8^3
# -------------------------------------------------------------
x3 = np.zeros(N, dtype=complex)
for k in range(4):
    a = x2[k]
    b = x2[k+4]
    w = W8**k
    x3[k]   = a + w*b
    x3[k+4] = a - w*b
print("\nStage 3 (final X):", np.round(x3, 4))

# -------------------------------------------------------------
# Compare to built-in FFT
# -------------------------------------------------------------
X_ref = np.fft.fft(x)
```

```
print("\nNumPy FFT:", np.round(X_ref, 4))
print("\nDifference:", np.round(x3 - X_ref, 8))
```

Input x[n] = [1.+0.j 0.+0.j 1.+0.j 0.+0.j -1.+0.j 0.+0.j -1.+0.j 0.+0.j]

Reorder: [1.+0.j -1.+0.j 1.+0.j -1.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]

Stage 1: [0.+0.j 2.+0.j 0.+0.j 2.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]

Stage 2: [0.+0.j 2.-2.j 0.+0.j 2.+2.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]

Stage 3 (final X): [0.+0.j 2.-2.j 0.+0.j 2.+2.j 0.+0.j 2.-2.j 0.+0.j 2.+2.j]

NumPy FFT: [0.+0.j 2.-2.j 0.+0.j 2.+2.j 0.+0.j 2.-2.j 0.+0.j 2.+2.j]

Difference: [0.+0.j 0.+0.j 0.+0.j -0.+0.j 0.+0.j 0.+0.j 0.+0.j -0.+0.j]

Python Tasks (run and discuss)

Example 2

Generate a random signal of length $N = 512$ and compare the speed of direct DFT versus FFT.

```
import numpy as np, time

N = 512
x = np.random.rand(N)

# Direct DFT
start = time.time()
X_dft = np.array([np.sum(x * np.exp(-2j*np.pi*k*np.arange(N)/N))
   for k in range(N)])
t_dft = time.time() - start

# FFT
start = time.time()
X_fft = np.fft.fft(x)
t_fft = time.time() - start

print(f"DFT time: {t_dft:.4f} s, FFT time: {t_fft:.6f} s, speedup
   : {t_dft/t_fft:.1f}x")
```

Visualize the magnitude spectrum and verify that both methods produce identical results.

```
plt.plot(np.abs(X_fft))
plt.title("Magnitude spectrum |X[k]|")
plt.show()
```

Example 2

Generate a random signal of length $N = 512$ and compare the speed of direct DFT versus FFT.

```
import numpy as np, time

N = 512
x = np.random.rand(N)

# Direct DFT
start = time.time()
X_dft = np.array([np.sum(x * np.exp(-2j*np.pi*k*np.arange(N)/N))
    for k in range(N)])
t_dft = time.time() - start

# FFT
start = time.time()
X_fft = np.fft.fft(x)
t_fft = time.time() - start

print(f"DFT time: {t_dft:.4f} s, FFT time: {t_fft:.6f} s, speedup
    : {t_dft/t_fft:.1f}x")
```

Visualize the magnitude spectrum and verify that both methods produce identical results.

```
plt.plot(np.abs(X_fft))
plt.title("Magnitude spectrum |X[k]|")
plt.show()
```

Example 3

Compare $O(N^2)$ direct DFT with $O(N \log N)$ FFT using experiment.

Tasks:

1. Implement the slow direct DFT.
2. Compare runtime of:
   - Direct DFT (nested sums),
   - FFT (`np.fft.fft`).
3. For $N = 64, 128, 256, 512$, measure and plot runtimes.
4. Explain how the experiment confirms the theoretical complexities.

Example code:

```python
def slow_dft(x):
    N = len(x)
    return np.array([np.sum(x * np.exp(-2j*np.pi*k*np.arange(N)/N
        )) for k in range(N)])

Ns = [64, 128, 256, 512]
t_dft, t_fft = [], []

for N in Ns:
    x = np.random.rand(N)
    start = time.time(); slow_dft(x); t_dft.append(time.time()-
        start)
    start = time.time(); np.fft.fft(x);   t_fft.append(time.time
        ()-start)

plt.loglog(Ns, t_dft, 'o-', label='DFT (O(N^2))')
plt.loglog(Ns, t_fft, 's-', label='FFT (O(N log N))')
plt.xlabel('N'); plt.ylabel('Time (s)'); plt.legend(); plt.show()
```

Example 4

Investigate how the execution time of the FFT depends on the signal length N, especially when N is or is not a power of two.

Tasks:

1. Test different signal lengths:

$$N \in \{500, 512, 1000, 1024, 2000, 2048\}.$$

2. For each N, generate a random signal and measure the execution time of `np.fft.fft()`.
3. Plot the FFT runtime as a function of N.
4. Discuss: Why are FFTs significantly faster when N is a power of two?

Example code:

```python
import numpy as np, time, matplotlib.pyplot as plt

Ns = [500, 512, 1000, 1024, 2000, 2048]
times = []

for N in Ns:
    x = np.random.rand(N)
    start = time.time()
    np.fft.fft(x)
    times.append(time.time() - start)
```

```
plt.plot(Ns, times, 'o-')
plt.xlabel('Signal length N')
plt.ylabel('FFT time (s)')
plt.title('FFT runtime vs. N')
plt.show()
```

Example 5

Investigate how numerical reconstruction error grows when computing FFT and inverse FFT.

Tasks:

1. For powers of two,
$$N = 2^k, \qquad k = 5, 6, \ldots, 12,$$
generate random signals.
2. Compute $y =$ `ifft(fft(x))` and measure the maximum absolute error
$$\max |x - y|.$$
3. Plot the reconstruction error versus N.
4. Discuss the result: Does the error grow linearly, quadratically, or logarithmically?

Example code:

```
Ns = 2**np.arange(5,13)  # 32..4096
errors = []

for N in Ns:
    x = np.random.rand(N)
    x_rec = np.fft.ifft(np.fft.fft(x))
    errors.append(np.max(np.abs(x - x_rec)))

plt.loglog(Ns, errors, 'o-')
plt.xlabel('N')
plt.ylabel('Max reconstruction error')
plt.title('Reconstruction error vs. N')
plt.show()
```